Process Agnostic Library Migration: Evaluation of Various Tools and Approaches

Joseph Murray
LiJun Li
ECE 4332 – Fall 2009
University of Virginia
jbm4t@virginia.edu
ll2bf@virginia.edu

ABSTRACT

Here we evaluate various tools and approaches to migration. The tools of PyCell and Cadence with SKILL are evaluated and the methods of Quality by Design (QbD) and Adaptive Migration with Constraint Relaxation are examined. We find that using Cadence with SKILL should be the preferred tool for this application due mainly to its greater degree of flexibility, its tool set of built-in migration functions and its integrated environment. In addition we propose a preferred migration algorithm that utilizes some of the QbD and the Adaptive Migration.

1 INTRODUCTION

As VSLI manufacturing processes continue to decrease in dimension to meet the expectations of Moore's Law, the process of migrating existing circuit topographies to new process generations has become increasingly difficult [4][5]. In the past, simple geometric shrinking and pitch adjustment was adequate for many circuit migrations [4]. However, this is not longer the case. Current process sizes require asymmetric shrinking of the various layers [6]. This has created a situation wherein the cost of migration has gone up significantly.

In addition, due to the high cost of and difficulty in obtaining rights to standard cell libraries of current generation processes, UVA lacks the ability to access these new technologies. This is a major limitation on research as it is desirable to prove designs in the most current processes. While new libraries for each new technology can be created by hand, the enormous effort of such an undertaking virtually eliminates this as a possibility. A better option might be to create a single process independent library and a protocol for easily porting the library objects to the desired technology. The first step in this process is to evaluate the tools and various approaches.

This Paper will give a brief introduction to PyCell and SKILL as well as the methods evaluated using these tools. A comparison will be made between the all the methods and tools used and finally recommendations will be made based on our findings for future work.

2 PYCELL

2.1 Introduction to PyCell

PyCell Studio is a parameterized cell design software developed by Ciranova Inc.. It is free to download and use, although one can pay to get commercial support from Ciranova. It is based on the popular programming language Python and implements OpenAccess as its default database file format. Hand-drawn layouts cannot be created in PyCell Studio instead one must write a script to specify the layout. The nature of this script is described below. More information about PyCell can be found at Ciranova's official website and our group homepage in which a PyCell tutorial has been created to demonstrate how to use PyCell in Linux without Ciranova's Python IDE.

2.2 Layout Creation

A Python script and a technology file containing all the design rules and parameters must be completed in order to create a PyCell layout. The Python script constructs a parameterized layout, as the template for instantiated cell in the future. The script constructs the layout by geometric considerations where each part of the layout is specified in relation to the other objects or a single reference point. Next, built-in PyCell API functions are used to retrieve the information in technology file, instead of being explicitly specified with exact numbers in the script. Once done, the script can be compiled to an OpenAccess database file together with technology file and PyCell API library. This OpenAccess file contains all the layout information of a standard cell and can be later read and viewed in the PyCell environment. The open cell can then be tested using the built-in design rule check (DRC) function. A well written Python script should automatically pass DRC. Otherwise one should go back to debug the Python script until it passes DRC. Figure 1 shows a flow chart depiction of the design process.

In this study, a parameterized inverter and a parameterized NAND were constructed in Python and ported to cni130 and cni180 technologies which are "fictious" but complete technologies that come with PyCell Studio for demonstration purpose. The generated results are in Figure 2. Typically a Python script of a parameterized inverter needs 300 lines while NAND (or NOR) needs about 350 lines.

2.3 Evaluation of PyCell Approach

The PyCell approach has several advantages. First, as long as the script is well written, it can be ported to most technologies without additional effort. Developers avoid being entangled with the numerical details of technology file but concentrate on the physical structure of standard cell. Since PyCell API also includes some advanced functions (especially those whose names begin with "fg") that can automate recognition and decision in terms of multiple design rules, it is easier for developers to design a parameterized cell with good portability.

However, there are also some disadvantages to this approach. The design rules may not be completely specified in migration. The Python script may need modification if new types of design rules in newer technologies have effects on the layout. Also, the investment to write a parameterized standard cell Python script can be considerable if the standard cells become complex. The writing and debugging of Python script takes up most of the developing time. In addition, a new script must be written for every new design. However, this is not as large of a hurdle as one might expect because code may be reused and as seen above

the code required does not sharply increase with complexity. Finally, apart from the commercial support from Ciranova, there is almost no community support one can refer to except the documentation of PyCell Studio when developing such a script.

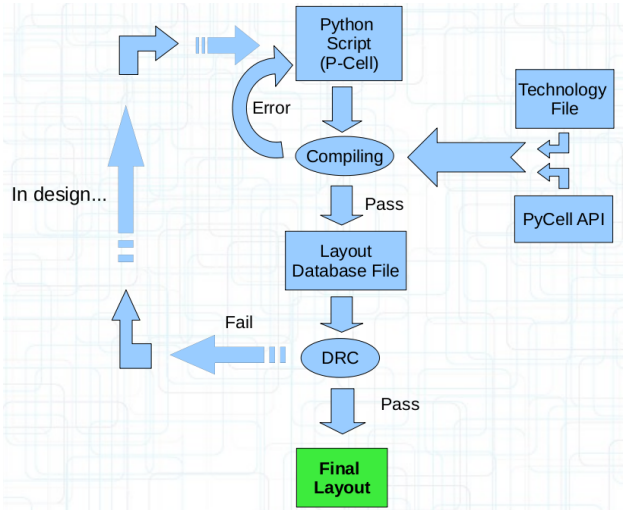
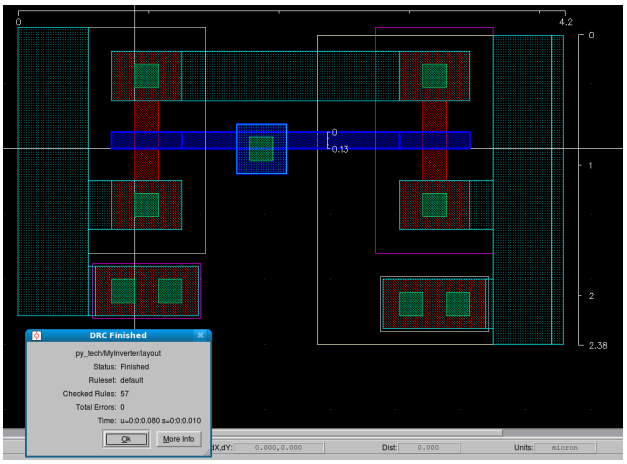
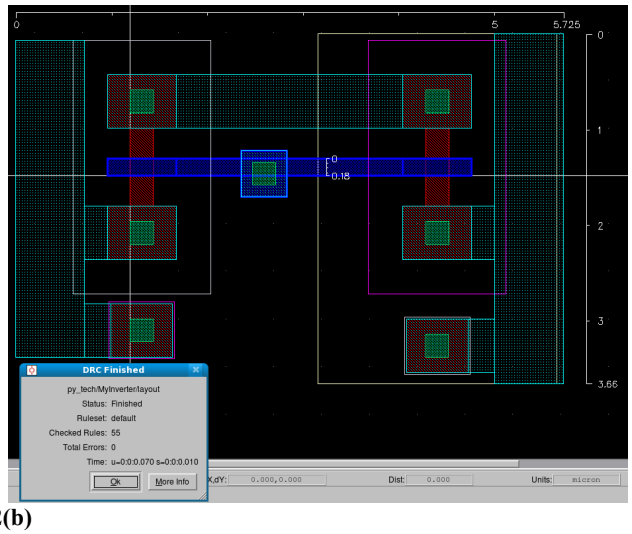


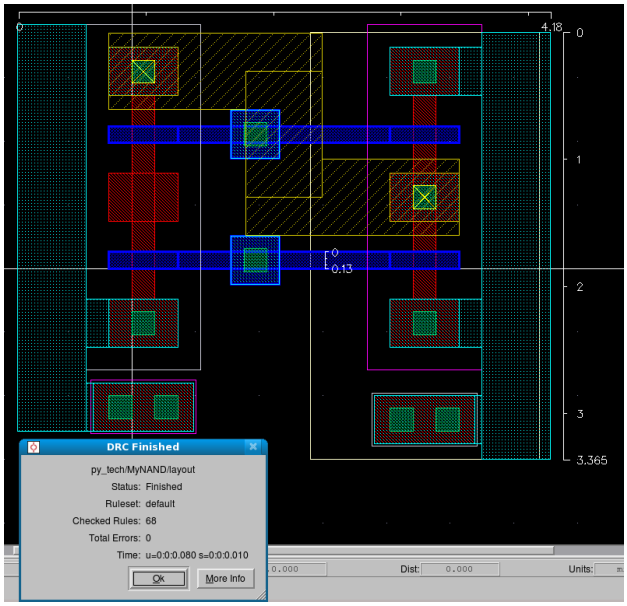
Figure 1: design flow of a parameterized cell in PyCell Studio.



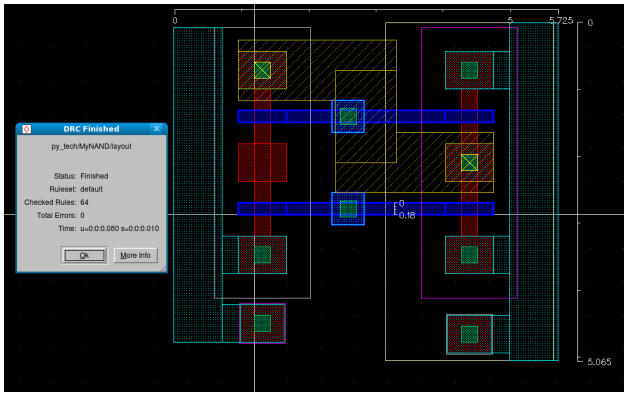
2(a)



2(b)



2(c)



2(d)

Figure 2: standard cells generated from two parameterized standard cell Python scripts using two different technologies. (a) an inverter under cni130 (130nm) technology. The left mosfet is nmos while the right is pnmos. The width of gate is 130 nm, and the x extent of inverter is 4.2 um while the y extent is 2.38 um. (b) an inverter under cni180 (180nm) technology with gate width of 180nm. The x extent is 5.725 um while the y extent is 3.66 um. (c) an NAND gate under 130nm technolog. The x extent is 4.2 um while the y extent is 3.365 um. (d) an NAND gate under 180 nm technology. The x extent is 5.725 um while the y extent is 5.065 um.

3 CADENCE WITH SKILL

3.1 Introduction to SKILL/Cadence

Cadence is a pay to use tool created by Cadence Design Systems, Inc. The tool encompasses a large number of sub-tools for design and testing of circuits. SKILL is a scripting language created for use in the Cadence environment and is based on the Lisp programming language. SKILL has many built in functions for data manipulation and formatting as would be expected but what makes it unique is its interface with the Cadence environment. SKILL has built in functions for every action allowed in the Cadence environment including opening and changing values on forms, manipulating schematic and layout objects, accessing library data, and many more. This makes the language

particularly powerful. While SKILL does share some similarities with other programming languages, learning SKILL and the specifics of the data structures used to store layout and technology file information proved to be a major portion of this study as no group members were previously familiar it.

There were a few aspects and features of SKILL/Cadence which were of particular interest to this study and should be addressed here. A Cadence layout is fully specified by a hierarchical data structure which contains information on the layers used, specifics of layout shapes, instances used, and so on. The other area of interest is the design rules. If a library containing a layout has a design rule file associated with it, details of this file are saved in another hierarchical structure. This structure contains data on layers available to this technology and rules for size and spacing of layout objects. SKILL also has several built in functions for extracting data from these data structures. For instance, the call: ***techGetSpacingRule(<pointer to techfile>, "minWidth", "poly")*** returns the minimum gate width. The last component that should be covered, is the shape structures which are part of the layout data structures. These contain all of the properties of a give layout object. This includes, shape type (e.g. rectangle), associated layer, ect.

More recently, Cadence has introduced a new language they have termed SKILL++. However, this study limited itself to an evaluation of SKILL.

3.2 Quality by Design Approaches

This approach is very similar to the approach used in PyCell. The difference is that, using the Cadence tool suite, layouts may be created first by hand. This suggests three ways of approaching the QbD approach. This first is to implement this exactly as in PyCell, where the entire layout is created based on a script. The script specifies size and position proportion relationships between the layout objects. The proportions are converted into actual numbers using values from the technology files. The second approach would be to use the built in tool for Cadence which allows the layout of P-Cells by hand. Here each layout object, size and position would be given a variable. A script would be used to fill in the values for each variable by reading the required values from the technology file. The third approach is to produce a fully specified layout by hand which is parsed by a script. The script would then change the size and position values of the layout objects based on values obtained from the technology file. The first approach requires a script for each cell to be ported. The later two methods require a layout to be created for each cell.

As a demonstration of the QbD approach, we chose to implement the third method described above. This was chosen because it was the easiest to implement (assuming a suitably restricted design space was used) and it contained aspects of all three approaches. The actual implementation of this demonstration was done by the following (See Figure 3 for flow chart depiction). The script accesses the file to be ported and the technology data structure associated with that file. The technology data is parsed to get the values needed for migration (here the gate size only was obtained). The file that will be used for output is then accessed, and the technology data for that file is then parsed. The script next changes the size of each layout shape based on ratios of the relevant data from the technology file. Here the objects were geometrically shrunk based on a ratio of the minimum gate sizes. The resultant shapes are then loaded into the output data structure. Finally, the script opens the output file, opens the form for design rule checking, and "presses" OK to run the design rule check. In this case the script must be redesigned if the design rule check fails as in the PyCell approach.

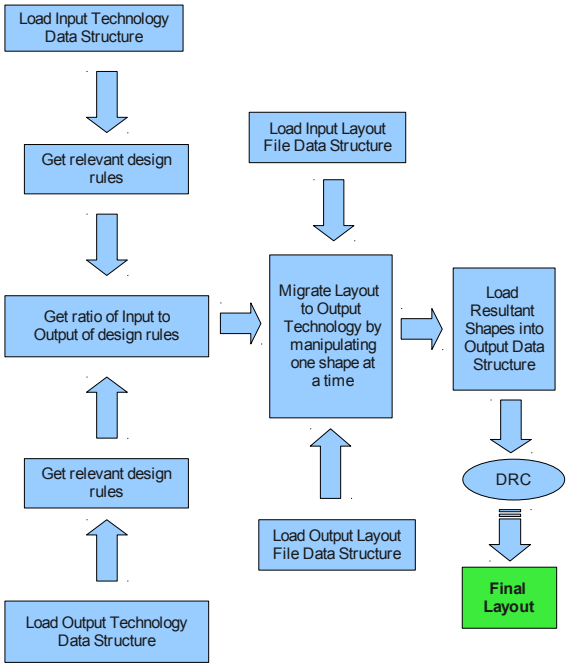


Figure 3: QbD migration algorithm flow chart depiction

While this is a simple script for this algorithm, it demonstrates the viability of several techniques. It shows that each of the other two QbD approaches described above could be implemented. First, this demonstrates the ability to access and manipulate layout data structures. This is fundamental to any approach that we might implement. Second, it shows that the technology data can easily be obtained. This is necessary for all of the QbD approaches. Figure 4 shows some results of a NOR gate migrated using this script. Inverter and NAND gates were also successfully ported but are not show here in the interest of space. This code was approximately 150 lines.

3.3 Evaluation of QbD Approach

The SKILL based QbD approach has essentially the same advantages and drawbacks of the PyCell approach.

3.4 Adaptive Migration and Constraint Relaxation Approach

This approach starts with a hand-drawn layout and migrates it to a new technology through a iterative process. After each step, the layout is checked or design rule errors. If design rule violations have occurred, position or size constraints may be relaxed. This means that the shape involved in the error may be moved or changed in size so that it is no longer in the same proportion to the rest of the layout as it was in the original layout. This can be a very complex problem. First, there needs to be a way to extract all the information needed on the errors. Second, there needs to be a reliable way of identifying which shapes are associated a given error. Depending on how the system saves the error, this may be anywhere from a simple data read to a full text parsing algorithm. Third, in order to correctly address the error in question the reason for the error then needs to be identified. Again this varies in complexity based on the system's error

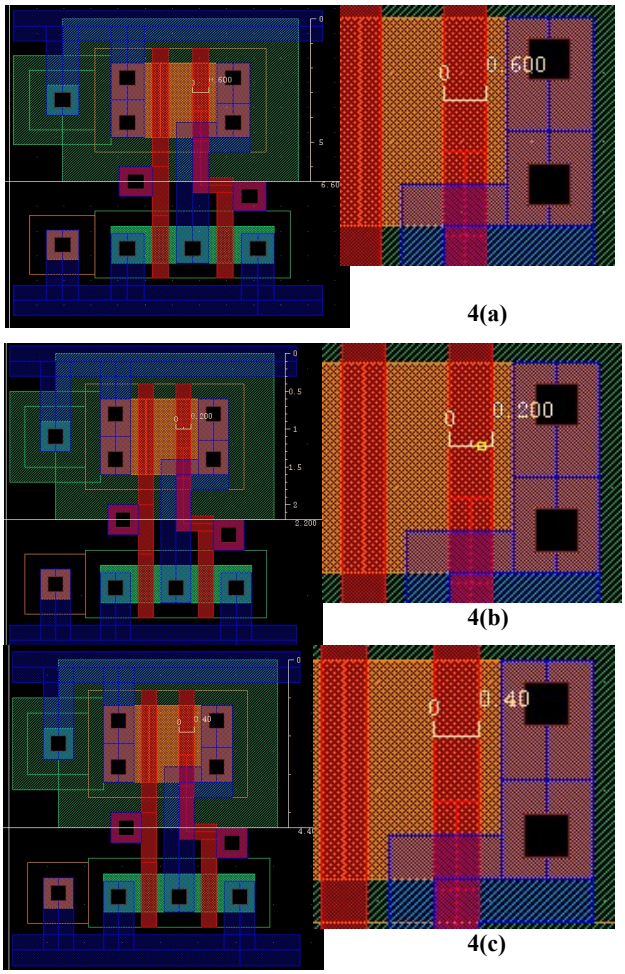


Figure 4: Examples of NOR gates migrated to various technologies using the QbD approach described above. Each frame shows the completed gate with DRC passed along with a close-up of the gates on the PMOS as confirmation of migration. Figure 4(a): original cell in the 600nm AMI process. Figure 4(b): cell from 4(a) ported to the 200nm AMI process. Figure 4(c): cell from 4(a) ported to the 400nm AMI process. Inverter and NAND gates were also successfully migrated but are not show here.

handling. Lastly, there needs to be a method for changing the proportion of a shape without effecting the rest of the layout. That means that in many cases, in order to repair a single error associated with a single shape, many other shapes would need to be altered.

Do to the great complexity of this problem, as a demonstration, of this method, we chose to deal with only a subset of the possible errors. The script we created only correctly repairs by constraint relaxation errors which are due to a shape being too small or large. The process works as follows (see Figure 5(a) for a flow chart depiction). The script first retrieves the data structures from the file to be ported and the file to be used for output. The iterative loop then begins. The output file is first cleared. Next, a migration factor (amount the layout will be shrunk or expanded by) for that step is defined. Then the shapes from the input data structure are migrated one at a time to the output data structure using the migration factor and DRC is run on the newly created layout. When DRC is run, it creates a series of shapes in the layout that represent the errors. These shapes can be accessed through the layout data structure and the shape data includes the coordinates of the error shape and a description of the error. Then the error parsing and constraint relaxation begins (see Figure 5(b) for a flow chart depiction.

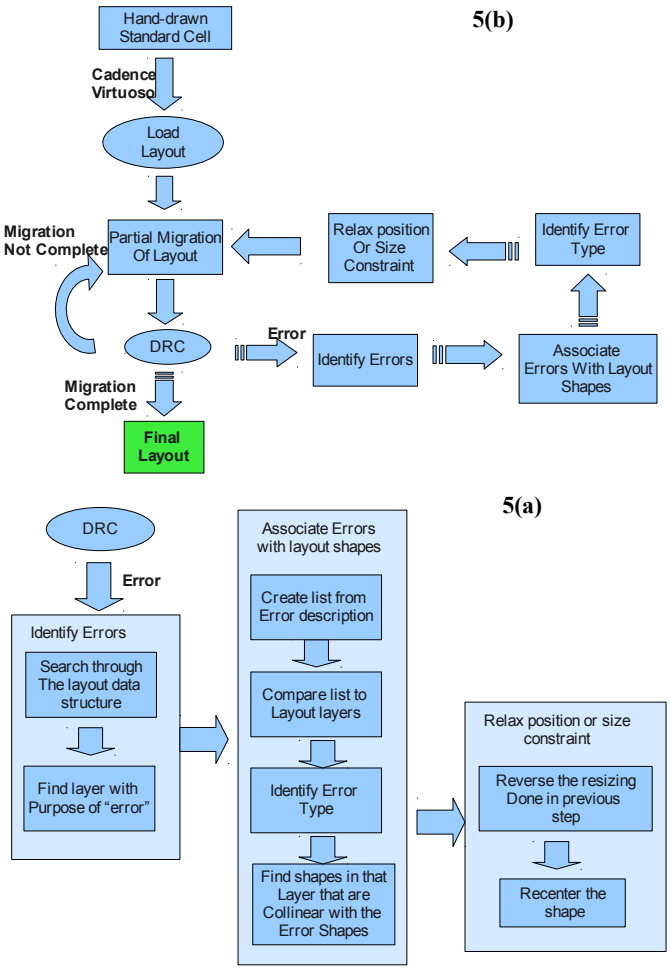
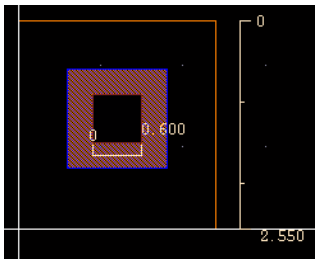


Figure 5(a): a flow chart depiction of the constraint relaxation method. Figure 5(b): a flow chart depiction of error handling protocol in the constraint relaxation method.

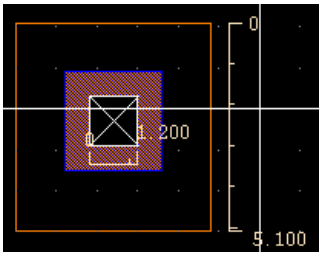
First the error layer in the layout data structure must be identified using the data from the layout structure. The actual error parsing then starts with finding the layer associated with the error. This is done by text parsing of the error description in the error shape data structure. A built in function creates a list of words in the description and each word is compared against the names of the layers in the layout data structure. The layer numbers are then saved. At this point, the error is also classified as either a sizing error, identified by having to do with one layer, or a inter layer spacing error. Single layer spacing errors were not dealt with here[1]. Next, all the coordinates of the error shape are decomposed into a series of lines (i.e. edges of a polygon). One at a time, the layout shapes on the same layers as the error are decomposed into lines. The script then compares each shape and error line for collinearity. If the shapes are collinear, it is assumed that they are associated with the error and a pointer to the shape is saved.

[1] In general, this requires more information to identify the error type and would be done later in the process.

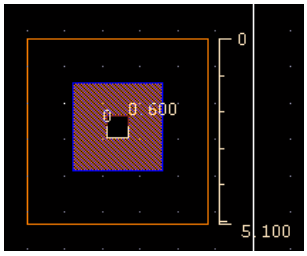
Next, the error is repaired. This is done simply by reversing the geometric shrinking and then re-centering the shape (this is where the assumption of a size constraint violation is used). Finally, the loop repeats itself until the cell is fully migrated. Figure 6 shows some results using this script. This script was approximately 500 lines of coding.



6(a)



6(b)



6(c)

Figure 6(a): A P-Tap created in a 600nm technology. Figure 6(b): A P-Tap migrated to a factious technology where the all parts are doubled in size except for vias. Here no adaptive migration is used and the layout fails DRC. Figure 6(c) the same P-Tap migration using adaption migration and constraint relaxation.

It should be noted here that there are a couple of additional assumptions used in the implementation of this method. First, it is assumed that the error description will contain a keyword(s) corresponding to a layer(s) in the layout. While this is not necessarily the case, this is not a limitation because a lookup table could simply be used if needed. Second, a number of assumptions are made about the error shapes, most important is collinearity with the shapes that caused the error. While we were not able to find the algorithm used to create these error shapes or documentation on this, we did find that it was implemented at the Cadence system level and was not technology specific. Thus it seems likely to be universal.

Even with these assumptions and with the reduced error handling scope, this demonstration shows the ability to address the first three of the four requirements of this method noted above. The fourth requirement however was only partially addressed in that no consideration was made for the effect the error repair might have on the rest of the layout. However, as shown above, manipulation of layout objects is no limitation in SKILL and is a algorithm issue only.

3.5 Evaluation of the Adaptive Migration and Constraint Relaxation Approach

This method had several advantages. First, once the algorithm is complete, no additional work is required to port any layout, which differs significantly from the PyCell approach. Also, this method adapts to new design rule requirements. That is to say, if there is a new design rule or a change in a design rule that could not be anticipated then this approach will handle it, unlike the QbD and PyCell approaches. Second, this is a subject with a rather mature literature, see [1], [2], [3], [5], and [6] just to name a few, meaning that a suitable algorithm may be adapted from others work. Third, this method may also be used for optimization of a given parameter (e.g. inter-wire capacitance or layout size) with relatively little additional work. All that would be needed is a way to evaluate a parameter because constraint relaxation functions will have already been developed. Finally, this approach is particularly applicable to the SKILL environment because SKILL has a set of built in functions for migration and constraint adjustment. However, we were not able to evaluate this due to a licensing issue that was not resolved before the end of the study.

There are some disadvantages to this method as well. This requires a large upfront investment in script development before it can be utilized for any useful migration[2]. Second, these algorithms are much more complex than the QbD approaches which only require a series of geometric considerations.

4 CONCLUSIONS: COMPARISON OF TOOLS AND METHODS

Table 1 shows a comparison of the tools tested here. From this study we found that there should be a significant preference for the use of Cadence with SKILL. Cadence should be the preferred tool for this application due mainly to its greater degree of flexibility, its tool set of built-in migration functions and its integrated environment. In addition, Cadence can perform all of the actions that my be performed in PyCell.

Table 1: Comparison of tools addressed in this study

Aspect	PyCell Studio	Cadence SKILL	Preference
Scripting Language	Python	SKILL (LISP-like)	Both well know. Depends on user.
Cost	Free to use but pay to get commercial support	Pay to use and get commercial support	PyCell preferred but UVA already has rights to Cadence
Migration support	Specifically designed for P-Cells	Special migration/ compaction tool set	Depends on preference for method
Extension	Almost none, have to interface with 3rd party	Good	Cadence strongly preferred
Community support	Almost none	Good	Cadence preferred
Emphasis	Parameterized cell	Generic	Cadence strongly preferred
Database	Open Access database	CDS Start supporting OA	No preference

Table 2 shows a comparison of methods for migration of standard cell libraries. Based on this data, we proposed a fourth option shown schematically in Figure 7. Here we suggest that in order

[2] This may change significantly with the use of the built in SKILL functions.

to reduce the complexity of the method of Adaptive Migration and Constraint Relaxation, some parts of the layout should be created using a QbD approach. This will allow some assumptions to be made about the layout during the DRC testing and error repair.

Table 2: Comparison of Methods Addressed in this Study

Aspect	PyCell	QbD	Constrain Relaxation
Up Front Cost	Depends on Library size/Complexity	Depends on Library size/Complexity	High
Maintenance Cost	Low	Low	Very Low
Cost to Extend to Optimization	Medium	Medium/Low	Very Low
Time to Initial Results	Low	Low	Very High
Complexity	Medium	Medium	High

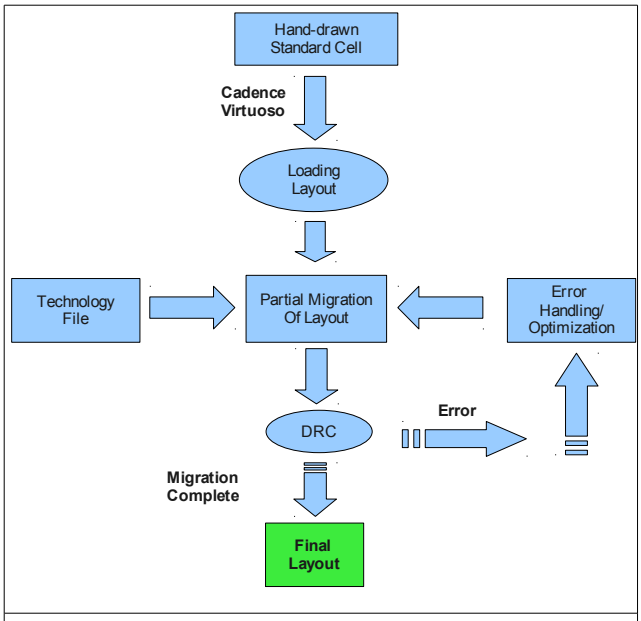


Figure 7: Proposed method of standard cell library migration.

5 REFERENCES

[1] Two-Dimensional Layout Migration by Soft Constraint Satisfaction, Proceedings of the Sixth International Symposium on Quality Electronic Design (ISQED'05), Q. Tang et. al.

[2]Calligrapher: A New Layout Migration Engine Based on Geometric Closeness,IEEE 2004,Fang Fang, Jianwen Zhu

[3]Timing-Driven Cell Layout De-Compaction for Yield Optimization by Critical Area Minimization, Proceedings of the conference on Design, automation and test in Europe: Proceedings, 2006 ,Tetsuya Iizuka, Makoto Ikeda, and Kunihiro Asada,

[4]Systems and Processes for Asymmetrically Shirking a VLSI Layout, US Patent, Kever et. al., 2006

[5]Two-Dimensional Layout Migration by Soft Constraint Satisfaction, Proceedings of the Sixth International Symposium on Quality Electronic Design (ISQED'05), Q. Tang et. al.

[6]Interconnect-driven Cell-based Migration of Integrated Circuit Layout, Evgeny Shaphir, 2009, Research Thesis